```c
/* Ce module définit les fonctions de manipulations des polynômes :
 * lecture, écriture, addition, soustraction, etc.
 *
 * @author: Vincent Granet "vg@unice.fr"
 * Creation @date:  18-Jan-2017 08:17
 * Last file update: 23-Jan-2017 08:33
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdarg.h>
#include <assert.h>
#include <math.h>

/*
 * Un polynôme est représenté par un tableau tel que
 * polynome[i] = coefficient de x^i
 */

#define DEGREMAX 40
typedef double polynome[DEGREMAX];

#define COEF(p,i) p[i]


/*****************************************************************************
*
 *
 *  Input
 *
 *****************************************************************************
*/

/*  Ce module fournit les fonctions de base d'entrée/sortie pour
 *  la gestion des polynômes
 *
 * @author: Vincent Granet "vg@unice.fr"
 * Creation @date:  8-Dec-2010 08:17
 * Last file update: 27-Dec-2010 16:02
 */

#include <stdio.h>
#include <ctype.h>

/*
 * Rôle : lit un entier sur l'entrée standard.
 * Conséquent : *c est le dernier caractère lu
 */
int lireEntier(int *c) {
  int neg = 0, n = 0;
  /* est-ce que l'entier est signé ? */
  if (*c=='+' || *c=='-') {
    if (*c=='-') neg=1;
    /* passer au caractère suivant */
    *c=getchar();
  }
  /* c est un chiffre ou la lettre x */
  if (*c=='x')
    return neg ? -1 : 1;
  /* lire l'entier */
  do
    n = 10*n + *c -'0';
  while (isdigit(*c=getchar()));
  return neg ? -n : n;
}
```



```c
/*
 * Rôle : sauter les espaces sur l'entrée standard
 * Conséquent : *c est le dernier caractère lu
 */
void sauterEspaces(int *c) {
  if (*c==' ')
    while ((*c=getchar()) == ' ');
}

/*
 * Rôle : retourne la réponse (un caractère) lu au clavier dans le menu
 *        de gestion du programme proposé à l'utilisateur
 */
char lireRep(void) {
  int c, rep = getchar();
  while ((c = getchar()) != '\n' && c != EOF) /* vide */;
  return rep;
}


/*
 * Rôle : écrit sur la sortie d'erreur standard le
 *        message d'erreur msg et arrête le programme
 */
static void erreur(char *msg) {
  fprintf(stderr,"erreur : %s\n", msg);
  exit(EXIT_FAILURE);
}

/* Rôle : renvoie le degré du polynôme p */
int degre(polynome p) {
  int d = DEGREMAX-1;
  while (d>=0) {
    if (COEF(p,d)!=0) return d;
    d--;
  }
  /* d==-1 =>  p = polynôme nul */
  return 0;
}

/*
 * Rôle : renvoie vrai si p1>=p2, faux sinon
 *        p1>=p2 => degré(p1)>=degré(p2)
 */
static int supEgal(polynome p1, polynome p2) {
  return degre(p1)>=degre(p2);
}

/* Rôle : teste si p(x)==0 */
static int zero(polynome p) {
  return degre(p)==0 && COEF(p,0)==0;
}

/* Rôle : diviser tous les coefficients de p par x  */
static void reduire(polynome p, double x) {
  for (int d=0; d<DEGREMAX; d++)
    if (COEF(p,d)!=0) COEF(p,d)/=x;
}

/*****************************************************************
 *                    Initialisation d'un polynôme               *
 *****************************************************************/

/*
 * Antécédent : nb égal le nombre de termes (coeff, degré) à
```

```
 *              initialiser
 *
 * Rôle : initialise le polynôme p avec les nb (coeff, degré) fournis
 *        en paramètre variable.
 *
 * Note : l'appel init(p,0), initialise p à 0
 */
void init(polynome p, int nb, ...) {
  va_list ap;

  /* initialiser  p à 0 */
  for (int i=0 ; i<DEGREMAX; i++) COEF(p,i) = 0;
  /* */
  va_start(ap, nb);
  while (nb-->0)
    COEF(p,va_arg(ap, int)) += va_arg(ap, double);
  /* */
  va_end(ap);
}

/*
 * Rôle : p1 = p2
 */
void copie(polynome p1, polynome p2) {
  for (int i=0 ; i<DEGREMAX; i++)  COEF(p1,i) = COEF(p2,i);
}

/*****************************************************************
 *                      Lecture d'un polynôme                    *
 *****************************************************************/

/*
 * Rôle : lit un monôme selon la syntaxe [[-|+][a]][x[^d]]
 * Conséquent : *c est le dernier caractère lu
 *              *coef est le coefficient du monôme lu
 *              *degré est le degré du monôme lu
 */
static void lireMonome(int *c, int *coef, int *degre) {
  /* sauter les éventuels espaces de tête */
  sauterEspaces(c);
  /* */
  *coef = 1;
  /* c est un chiffre ou + ou -*/
  if (isdigit(*c) || *c == '-' || *c == '+')
    /* lire le coefficient */
    *coef = lireEntier(c);
  if (*c!='x')
    /* pas d'inconnue */
    *degre = 0;
  else
    /* lire l'inconnue et le degré */
    if ((*c=getchar()) == '^') {
      *c = getchar();
      /* lire le degré */
      *degre = lireEntier(c);
    }
    else
      *degre = 1;
}

/*
 * Rôle : lit un polynôme sur l'entrée standard
 * Conséquent : p est le polynôme lu
 * Note : la fin de ligne termine le polynôme
 */
void lire(polynome p) {
```



```
  int neg= 0, c, coef, degre;

  init(p,0);
  while ((c = getchar())) {
    /* lire le prochain monôme */
    lireMonome(&c, &coef, &degre);
    /* l'ajouter au polynôme p */
    if (degre>=DEGREMAX)
      erreur("degré trop grand");

    COEF(p,degre)+= neg ? -coef : coef ;
    sauterEspaces(&c);
    if (c=='\n')
      /* on a terminé la lecture du polynôme */
      return;
    /* else signe + ou - attendu */
    if (c!='+' && c!='-')
      erreur("+ ou - attendu");

    neg = (c=='-');
  }
}

/*****************************************************************
 *                      Écriture d'un polynôme                   *
 *****************************************************************/

/*
 * Antécédent : coef != 0, signePlus==1 si un plus doit être
 *              mis devant un coef positif
 * Rôle : écrit le monôme (coef,degre) sur la sortie standard
 *        sous la forme normalisée
 */
static void ecrireMonome(double coef, int degre, int signePlus) {
  assert(coef!=0);
  if ((coef==1 || coef==-1) && degre!=0) {
    /* on n'écrit pas le coefficient devant un x */
    if (coef==-1) printf("-");
    else // coef==1
      if (signePlus) printf("+");
  }
  else
    if (signePlus)
      /* on écrit le coefficient avec le signe + ou - */
      printf(((int) coef) == coef ? "%+.0f" : "%+.2f", coef);
    else
      printf(((int) coef) == coef ? "%.0f" : "%.2f", coef);
  /* */
  if (degre>0) {
    /* écrire l'inconnue */
    printf("x");
    if (degre>1)
      /* écrire le degré */
      printf("^%d", degre);
  }
}

/*
 * Rôle : écrit sur la sortie standard le polynôme sous forme
 *        normalisée en partant de son degré le plus élevé
 */
void ecrire(polynome p) {
  int d = degre(p);

  if (d==0 && COEF(p,d)==0)
    /* cas particulier : polynôme nul */
```

```
    printf("0");
  else {
    /* écrire le premier monôme (ne pas mettre un éventuel + initial) */
    ecrireMonome(COEF(p,d), d, 0);
    for (d--; d>=0; d--)
      if (COEF(p,d)!=0)
        ecrireMonome(COEF(p,d), d, 1);
  }
}

/*
 * Rôle : écrit sur la sortie standard le polynome suivi d'un \n
 */
void ecrireln(polynome p) {
  ecrire(p); putchar('\n');
}

/*****************************************************************************
 *                        Fonctions de manipulation des polynômes            *
 *****************************************************************************/

/*
 * Antécédent : p1 et p2 deux polynômes initialisés
 * Conséquent : p3 = p1 + p2
 */
void additionner(polynome p1, polynome p2, polynome p3) {
  for (int d=0; d<DEGREMAX; d++) COEF(p3,d) = COEF(p1,d)+COEF(p2,d);
}

/*
 * Antécédent : p1 et p2 deux polynômes initialisés
 * Conséquent : p3 = p1 - p2
 */
void soustraire(polynome p1, polynome p2, polynome p3) {
  for (int d=0; d<DEGREMAX; d++) COEF(p3,d) = COEF(p1,d)-COEF(p2,d);
}

/*
 * Antécédent : p1 et p2 deux polynômes initialisés
 * Conséquent : p3 = p1 * p2
 * Note       : degré(p3) = degré(p1) + degré(p2)
 */
void multiplier(polynome p1, polynome p2, polynome p3) {
  init(p3,0);
  if (!zero(p1) && !zero(p2)) {
    /* faire le produit de tous les monômes (!=0) de p1
     * par p2 et les additionner
     */
    if (degre(p1)+degre(p2)>=DEGREMAX)
      erreur("multiplication : overflow");
    // else
    for (int d1=0 ; d1<DEGREMAX; d1++)
      for(int d2=0;d2<DEGREMAX; d2++)
        if (COEF(p1,d1)!=0 && COEF(p2,d2)!=0)
          COEF(p3,d1+d2)+=COEF(p1,d1)*COEF(p2,d2);
  }
}

/*
 * Antécédent : p polynôme initialisé
 * Conséquent : q  = p'
 */
void deriver(polynome p, polynome q) {
  init(q,0);
  for (int d=DEGREMAX-1 ; d>0; d--)
    if (COEF(p,d)!=0)
```



```
      COEF(q,d-1) = COEF(p,d)*d;
}

/*
 * Rôle : renvoie l'évaluation de p(x)
 * Note : Algorithme d'Horner
 */
double evaluer(polynome p, double x) {
  int n = DEGREMAX-1;
  double valeur = COEF(p,n);
  /* valeur = \sum\inff{k=n}\supp{n}{p[k]\,x\supp{k-1}} = p[n] */
  for (int d=n-1; d >= 0; d--) {
    /* valeur = \sum\inff{k=n}\supp{d}{p[k]\,x\supp{k-1}} */
    valeur = valeur*x + COEF(p,d);
  }
  /* valeur = \sum\inff{k=n}\supp{0}{p[k]\,x\supp{k}} */
  return valeur;
}


/*
 * Antécédent : a et b deux polynômes initialisés b!=0
 * Conséquent : a = b*q + r, degré(r)<degré(b)
 * Rôle : calcule la division euclidienne de a par b
 */
void diviser(polynome a, polynome b, polynome q,  polynome r) {
  if (zero(b)) erreur("Division par 0");
  /* Ok on peut procèder à la division */
  int degreB = degre(b);
  if (degreB==0) {
    /* r = 0; et q = a/COEF(b,0) */
    init(r,0);
    copie(q,a);
    reduire(q,COEF(b,0));
  }
  else {
    init(q,0); /* q = 0 */
    copie(r,a); /* r = a */
    while (supEgal(r,b)) {
      /*  chercher par combien de fois il faut multiplier le
       *        degre max de B pour obtenir le degré de r
       */
      int degreR = degre(r);
      int degreQ = degreR-degreB;
      double coefQ = COEF(r,degreR)/COEF(b,degreB);
      polynome aux, m;

      init(m, 1, degreQ, coefQ);
      multiplier(m, b, aux);
      soustraire(r, aux, r);
      additionner(m,q,q);
    }
  }
}

/*
 * Rôle : factorise p1 par (x-alpha). Renvoie 1 si p1 est factorisable,
 *        et p2 est le polynôme factorisé ; sinon renvoie 0
 */
int factoriser(polynome p1, double alpha, polynome p2) {
  /* si p1(x) est factorisable par (x-alpha) alors alpha est racine de
   * p1(x) et donc p1(alpha) = 0
   */
  if (evaluer(p1,alpha)!=0)
    // alpha n'est pas racine de p1
    return 0;
```

```
  // else p1 est factorisable
  polynome q, r;
  init(q, 2, 1, 1.0, 0, -alpha);
  diviser(p1, q, p2, r);
  return 1;
}

/*
 * Antécédent : p2 non nul
 * Rôle : p3 = pgcd(p1, p2)
 *        selon l'algorithme d'Euclide
 */
void pgcd(polynome p1, polynome p2, polynome p3) {
  polynome a, b;
  assert(!zero(p2));
  //
  if (supEgal(p1,p2)) {
    copie(a,p1);
    copie(b,p2);
  }
  else {
    // p2>p1
    copie(a,p2);
    copie(b,p1);
  }
  // R0=a, R1=b et a>=b
  // on calcule la suite Rn-1 = Rn*Qn + Rn+1
  // où les Rn+1 sont le reste de la division euclidienne
  // de Rn_1 par Rn. Lorsque Rn+1 est égal à 0, on a alors
  // le Rn = pgcd(a,b)
  while (!zero(b)) {
    polynome q, r;
    diviser(a, b, q, r);
    copie(a,b);
    copie(b,r);
  }
  // rendre a unitaire => diviser tous les coeff de a par le coeff du degre max
a a
  // de telle façon que le coeff du degre max soit égal à 1
  reduire(a, COEF(a,degre(a)));
  // a est le pgcd(a,b)
  copie(p3,a);
}

/*
 * Rôle : p3 = ppcm(p1, p2) = (p1*p2)/pgcd(p1,p2)
 */
void ppcm(polynome p1, polynome p2, polynome p3) {
  if (zero(p2))
    // par cnovention
    init(p3,0);
  else {
    polynome prod, pegecede, r;
    // calculer le produit p1*p2
    multiplier(p1, p2, prod);
     // calculer pgcd(p1,p2)
    pgcd(p1,p2,pegecede);
    //  ppcm(p1,p2) = (p1*p2)/pgcd(p1,p2)
    diviser(prod, pegecede, p3, r);
    // rendre p3 unitaire
    reduire(p3, COEF(p3,degre(p3)));
  }
}

/*****************************************************************************
*
```



```
 *
 *  Interface utilisateur
 *
 *****************************************************************************
*/

/*
 * Rôle : lit un polynôme et un réel sur l'entrée standard, évalue le
 *        polynôme et écrit le résultat sur la sortie standard
 */
void evaluation(void) {
  polynome p;
  double x;

  printf("p(x) = "); lire(p);
  printf("x = "); scanf("%lf",&x);
  printf("\np(%f) = %f\n", x, evaluer(p, x));
}

/*
 * Rôle : lit un polynôme sur l'entrée standard, et écrit le polynôme
 *        dérivé sur la sortie standard
 */
void derivation(void) {
  polynome p1, p2;

  printf("p(x) = "); lire(p1);
  deriver(p1,p2);
  printf("p'(x) = "); ecrireln(p2);
}

/*
 *  Antécédent : op est égal à '+', '-' ou '*'
 *  Rôle : lit 2 polynômes sur l'entrée standard, exécute l'opération
 *         op et écrire le polynôme résultat sur la sortie standard
 */
void AjSousMult(char op) {
  polynome p1, p2, p3;

  printf("p1(x) = "); lire(p1);
  printf("p2(x) = "); lire(p2);
  switch (op) {
   case '+' : additionner(p1, p2, p3); break;
   case '-' : soustraire(p1, p2, p3); break;
   case '*' : multiplier(p1, p2, p3); break;
  }
  printf("p1(x) %c p2(x) = ", op);
  ecrireln(p3);
}

/*
 *  Rôle : lit 2 polynômes sur l'entrée standard, et écrit sur la
 *         sortie standard le quotient et le reste de la division
 *         euclidienne des deux polynômes
 */
void division(void) {
  polynome p1, p2, q, r;

  printf("p1(x) = "); lire(p1);
  printf("p2(x) = "); lire(p2);
  diviser(p1, p2, q, r);
  printf("quotient = "); ecrireln(q);
  printf("reste = "); ecrireln(r);
}

void plusgrandcommundiviseur() {
```

```
  polynome p1, p2, gcd;
  printf("p1(x) = "); lire(p1);
  printf("p2(x) = "); lire(p2);
  pgcd(p1, p2, gcd);
  printf("pgcd = ");  ecrireln(gcd);
}

void pluspetitcommunmultipe() {
  polynome p1, p2, res;
  printf("p1(x) = "); lire(p1);
  printf("p2(x) = "); lire(p2);
  ppcm(p1, p2, res);
  printf("ppcm = ");  ecrireln(res);
}

/*
 *  Rôle : lit un polynôme et un réel alpha sur l'entrée standard, et,
 *         si cette factorisation est possible (ie alpha racine du
 *         polynôme), écrit sur la sortie standard le résultat de la
 *         factorisation du polynôme par (x-alpha),
 */
void factorisation(void) {
  polynome p1, p2;
  double alpha;

  printf("p(x) = "); lire(p1);
  printf("alpha = "); scanf("%lf",&alpha);
  if (factoriser(p1, alpha, p2)) {
    char signe = alpha<0 ? '+' : '-';
    printf("p(x) = (x%c%.2f)(", signe, fabs(alpha));
    ecrire(p2); printf(")\n");
  }
  else
    fprintf(stderr,"polynôme non factorisable\n");
}

/*********************************************************************
 *                        Le programme principal                     *
 *********************************************************************/

int main(void) {
  char rep;

  printf("Gestion de polynômes :\n");
  while  (1) {
    printf("\n-----------------------------------\n");
    printf("\t+ : addition\n");
    printf("\t- : soustraction\n");
    printf("\t* : multiplication\n");
    printf("\t/ : division\n");
    printf("\td : dérivation\n");
    printf("\te : évaluation\n");
    printf("\tf : factorisation par (x-a)\n");
    printf("\tp : pgcd\n");
    printf("\ts : ppcm\n");
    printf("\tq : quit\n\n");
    printf("cmd > ");
    /* lire la commande de l'utilisateur et l'exécuter */
    switch (rep=lireRep()) {
      case '+' :
      case '-' :
      case '*' : AjSousMult(rep); break;
      case '/' : division(); break;
      case 'd' : derivation(); break;
      case 'e' : evaluation(); break;
      case 'f' : factorisation(); break;
```



```
      case 'p' : plusgrandcommundiviseur(); break;
      case 's' : pluspetitcommunmultipe(); break;
      case 'q' : return EXIT_SUCCESS;
      default :  fprintf(stderr, "commande '%c' inconnue\n",rep);
    }
  }
}
```